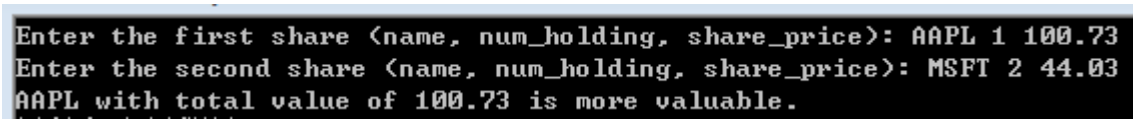
Review: Using an `istringstream` for String Parsing

An `istringstream` is often used when we **NEED** to do some work with **individual words within a string**.

The `istringstream` type reads a `string`. The characters in the string sequence can be extracted from the `istringstream` object using any operation allowed on input streams (e.g., >>).

題型 A: `string` 的內容與格式單純，我們可以簡單用 `istringstream` 解析出需要的資訊

In-class Exercise 8.1: Input two stocks with each stock containing stock name, number of share owned and the share price and report the most valuable one. For example:



The client code looks like:

StockClient.cpp

```
#include <iostream>
#include "Stock.h"

using namespace std;

int main()
{
    cout << "Enter the first share (name, num_holding, share_price): ";
    string line;
    getline(cin, line);
    Stock s0(line);

    cout << "Enter the second share (name, num_holding, share_price): ";
    getline(cin, line);
    Stock s1(line);
    Stock top = s0.topval(s1);
    cout << top.getName() << " with total value of "
       << top.getTotalVal() << " is more valuable." << endl;
    return 0;
}
```

And `Stock.h` looks like:

Stock.h

```
#ifndef STOCK_H
#define STOCK_H
#include <string>

class Stock
{
```

```
public:
    Stock(std::string record);
    const Stock& topval(const Stock& s) const;
    std::string getName() const {return name;}
    double getTotalVal() const {return total_val;}
private:
    std::string name;
    unsigned share_num;
    double share_val;
    double total_val;
};

#endif // STOCK_H
```

Implement the `Stock.cpp`.

A:

```
#include "Stock.h"
#include <string>
#include <sstream>
using namespace std;

Stock::Stock(string record)
{
    istringstream input_istring(record);
    input_istring >> name >> share_num >> share_val;
    total_val = share_num * share_val;
}

const Stock& Stock::topval(const Stock& s) const
{
    if (s.total_val > total_val) return s;
    else return *this;
}
```

題型 B: `string` 的內容與格式不單純，我們需先整理 `string`，才可以簡單用 `istringstream` 解析出需要的資訊

The MATLAB-like input syntax discards white space and uses semi-colon to separate different rows and comma to separate the elements in a row. The syntax goes like:

```
A=[1.1, 3.0, 6.5; 7.8, 4.5, 2.2; 3.4, 5.6, 8.9]
```

We would like to put these `double` elements into a matrix `vector<vector<double>>`.

解題技巧：我們需先處理此 `string` before using `istringstream`. 我們第一步先把

elements in a row separated by semicolon 解析出，並將資訊放在 `vector<string>`。

```
A=[1.1, 3.0, 6.5; 7.8, 4.5, 2.2; 3.4, 5.6, 8.9]
```

```
string A;
getline(cin, A); // A=[1.1, 3.0, 6.5; 7.8, 4.5, 2.2; 3.4, 5.6, 8.9]
vector<string> vs;
string s;
for (auto c : A) {
    if (c == '[') {
        s.clear();
    }
    else if (c == ',')
        s.push_back(' ');
    else if (c == ';' || c == ']'){
        vs.push_back(s);
        s.clear();
    }
    else
        s.push_back(c);
}
```

Q: If we print the contents of `vector<string> vs` after parsing:

```
for (auto e : vs)
    cout << e << endl;
```

What are the outputs if we input a string?

```
A=[1.1, 3.0, 6.5; 7.8, 4.5, 2.2; 3.4, 5.6, 8.9]
```

A:

第二步再解析出 each element in a row. 因為 `vector<string> vs` 的每一 `string` 的內容與格式單純，我們可以簡單用 `istringstream` 解析出需要的資訊。